VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Automatická klasifikace textu vzhledem k období

Automatic Text Classification Relative to the Period

2020

Jakub Chynoradský

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Chynoradský**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Automatická klasifikace textu vzhledem k období
Automatic Text Classification Relative to the Period

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je ověřit algoritmy pro zpracování textu pro datování textů do správného období a případně přiřazení autorů. Během práce bude otestováno několik metod pro modelování textu a využití těchto modelů pro správnou dataci.

Práce bude obsahovat:
1. Popis existujících řešení a řešení podobných problémů.
2. Popis metod využitelných pro datování textů.
3. Vytvoření kolekce dat pro experimenty.
4. Implementace několika vybraných metod.
5. Ověření zvolených metod a vyhodnocení výsledků.

Program bude implementován v jazyce Python nebo jiném vhodném.

Seznam doporučené odborné literatury:

[1] AGGARWAL, Charu C. Machine learning for text. New York, NY: Springer Science+Business Media, 2018. ISBN 978-3-319-73530-6.
[2] AGGARWAL, Charu C. Data mining: the textbook. New York, NY: Springer Science+Business Media, 2015. ISBN 978-3-319-14141-1.
[3] HASTIE, Trevor., Robert. TIBSHIRANI a J. H. FRIEDMAN. The elements of statistical learning: data mining, inference, and prediction. 2nd ed. New York, NY: Springer, c2009. ISBN 978-0-387-84858-7.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Jan Platoš, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal

V Ostravě 15. května 2020 .....

Rád bych poděkoval vedoucímu mé práce doc. Ing. Janu Platošovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této bakalářské práce.

Abstrakt

Cílem této bakalářské práce bylo vybrat a otestovat několik metod pro datování textů do správného období. Účelem těchto metod je zařadit větu historického textu do jedné ze čtyř kategorií od 17. do 20. století. Mnou zvolené metody všechny spadají do kategorie strojového učení, proto byla součástí práce také tvorba rozsáhlého vzorku dat pro učení a testování modelů. Pro implementaci byl využit objektově orientovaný programovací jazyk Python.

Klíčová slova: datování textů, strojové učení, hluboké učení, náhodný les, klasifikace, přenesené učení, python

Abstract

The aim of this bachelor thesis was to select and test several methods for dating texts to the correct period. The purpose of these methods is to classify a sentence to one of four categories from the 17th century to the 20th century. The methods I have chosen all belong to the machine learning category, which is why my thesis included an extensive data sample for learning and testing models. An object-oriented programming language Python was used for implementation.

Keywords: dating texts, machine learning, deep learning, random forest, classification, transfer learning, python

Obsah

Seznam použitých zkratek a symbolů

Dataset	–	kolekce dat použitých pro experimenty
CPU	–	centrální procesorová jednotka (procesor)
GPU	–	grafická procesorová jednotka (grafická karta)

Seznam obrázků

Seznam tabulek

1 Úvod

Ať už se jedná o chemicko-fyzikální nebo počítačové datování starých textů, tak datování hraje velmi důležitou roli v úplném porozumění informací, které historické texty sdělují. V mnohých starých textech nám není znám ani autor. Zjištění správného data nám například může pomoci si propojit důležité středověké bitvy a zjistit tak o nich více, což je klíčové pro mnohé obory zabývající se zkoumáním naší historie.

S narůstajícím výpočetním výkonem se nám kromě tradičních lingvistických metod odemykají nové cesty, jak datovat text. Jedna z nejpopulárnějších metod pro hledání určitých vzorů v textu je cesta strojového učení. Na základě tisíce vstupních vzorků již známých textů propojených s daty je strojové učení schopno automaticky najít vzorce mezi vstupními daty a s velkou přesností provádět požadované úkoly.

Tato bakalářská práce se zabývá automatickým zařazením textu do historického období. Je tedy potřeba vytvořit klasifikační systém, který zařadí věty nebo odstavec textu do správného období. Jako klasifikační kategorie jsou v této práci zvolené čtyři století.

První část práce se věnuje teoretickým základům v oblasti strojového učení a klasifikace a popisu metod, které jsou následně v práci využity pro tvorbu modelů pro datování. Praktická část práce začíná kapitolou o shromáždění potřebného vzorku dat. A následně se práce přesune na implementaci tří dříve popsaných metod pro správnou dataci textu, které všechny patří do kategorie strojového učení. Poslední část je věnována shrnutí výsledků jednotlivých metod a celkové vyhodnocení výsledků.

2 Přístupy pro tvorbu klasifikačního systému

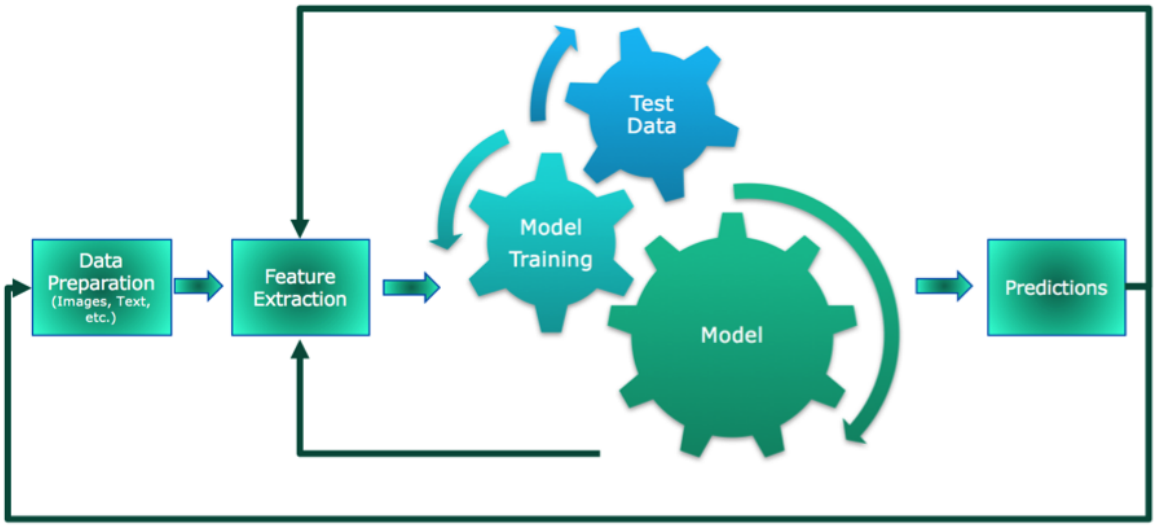
2.1 Systém založen na pravidlech (Rule-Based System)

Jedná se o systém, který aplikuje ručně vytvořená pravidla pro kategorizaci dat. Snaží se napodobit lidskou inteligenci. Proto aby správně fungoval je potřeba zdroj dat a sadu pravidel pro manipulaci s daty. Těmto pravidlům se také někdy říká “tvrzení pokud”, protože mají většinou tvar “pokud se stane X, tak proveď Y”. Jednim z hlavních nedostatků je to, že pokud jsou podmínky zvolené špatně nebo vynechané, tak systém nebude pracovat správně. [1, 2]

Při datování textů jsou využívané různé lingvistické prostředky jako například rozpoznávání dialektů nebo použitých archaismů v textech.

2.2 Strojové učení (Machine learning)

Strojové učení je skupina metod datové analýzy, která automatizuje tvorbu analytického modelu a je považována za podskupinu umělé inteligence. Strojové učení je schopné se samostatně učit a zdokonalovat bez toho, aby bylo přímo naprogramováno. K tomu je třeba strojovému učení dodat vhodný datový vzorek, takže místo spoléhání se na ručně vytvořená pravidla si zde metoda sama najde vzorce, na základě kterých bude vykonávat požadovanou činnost. Modely jdou tímto způsobem vytrénovat na různé činnosti, jako například filtrace nežádoucích emailů, rozpoznávání obličejů nebo třeba shrnutí internetových článků. Při klasifikaci s dozorem, kterou budeme využívat, je potřeba aby datový vzorek obsahoval ke každému záznamu i příslušnou skupinu. V dnešní době nachází strojové učení široké využití a dokonce v takových odvětvích jako například zdravotní, finanční nebo právní sektor. [3, 4]



Obrázek 1: Příklad procesu tvorby modelu strojového učení [5]

Pro případ klasifikace se strojového učení se dělí do dvou hlavních kategorií:

2.2.1 Klasifikace s dozorem (Supervised Classification)

Klasifikace je proces výběru správné kategorie pro poskytnutý vstup. Při klasifikaci s dozorem jsou kategorie, ze kterých se vybírá předem definované. Příkladem může být automatické rozdělení novinových článků do předem definovaných kategorií jako sport, politika atd. [6]

Klasifikaci s dozorem můžeme dále dělit na 2 hlavní kategorie:

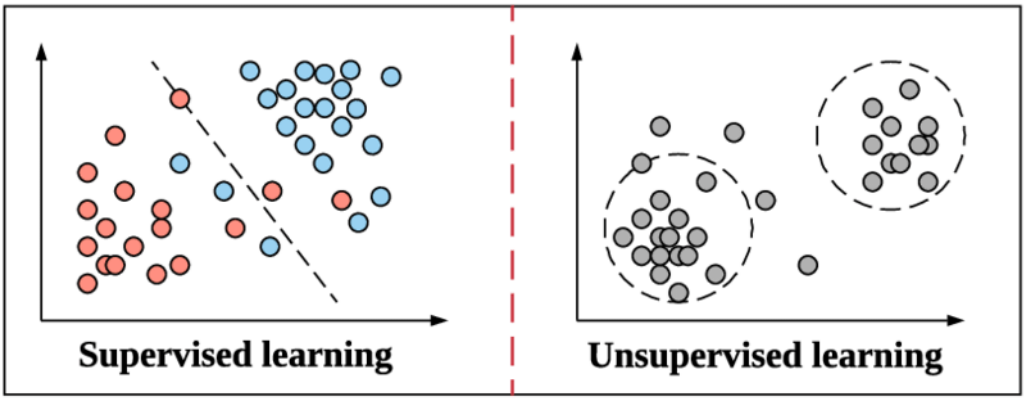
- Klasifikace (Classification) – Využívá se, pokud výstupem jsou kategorie, jako například: „sport“ a „politika“.
- Regrese (Regression) – Využívá se, pokud výstupem jsou reální hodnoty, jako například: „výška“ a „váha“.

2.2.2 Klasifikace bez dozoru (Unsupervised Classification)

Opakem klasifikace s dozorem je klasifikace bez dozoru, kde se vstup řadí do předem neznámých konkrétních kategorií nebo témat podle určitých opakujících se vzorů v daných vstupech. Cílem je tedy navíc vytvoření struktury pro klasifikaci. [6]

Klasifikaci bez dozoru můžeme dále dělit na 2 hlavní kategorie:

- Shlukování (Clustering) – Využívá se, pokud chceme v datech najít vlastní seskupení, jako například: seskupení zákazníků podle nákupního chování.
- Asociace (Associations) – Využívá se, pokud chceme v datech najít pravidla, které popisují velké části vašich dat, jako například: lidé, kteří kupují výrobek X často kupují i výrobek Y.



Obrázek 2: Rozdílu mezi klasifikací s dozorem a bez dozoru [7]

2.3 Hybridní systém (Hybrid system)

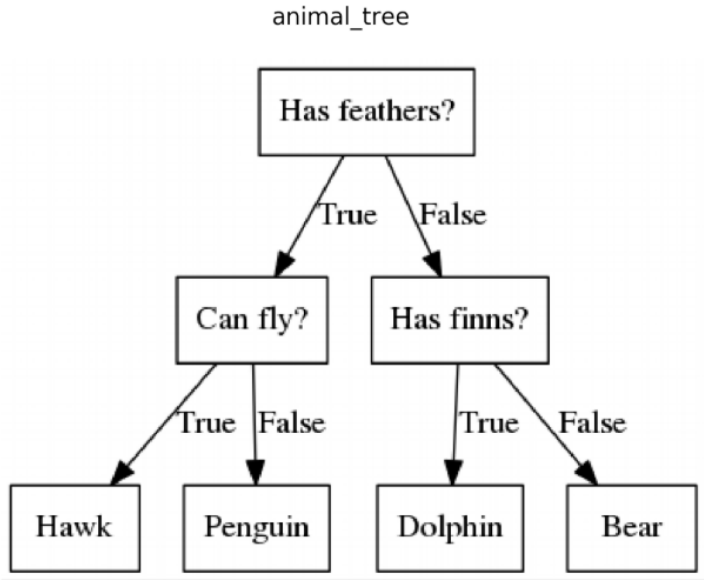
Hybridní systémy jsou systémy, které jsou kombinací systému založeném na pravidlech a strojového učení. Základem takových systému je model strojového učení, který je následně vylepšen o ručně psaná pravidla. Tato vytvořené systémy můžou pomocí podmínek vylepšit chyby strojového učení a dále vylepšit jeho přesnost. [8]

3 Metody využitelné pro datování textů

3.1 Náhodný les (Random Forest)

3.1.1 Rozhodovací strom (Decision Tree)

Abychom pochopili metodu učení 'Náhodný les' musíme nejdříve pochopit základní stavební kámen této metody. Rozhodovací strom je, jak už jeho název napovídá, model, který vytváří stromovou strukturu podmínek, které se naučí pomocí testovacích dat rozdělených do kategorií. Na základě této stromové hierarchie potom při přijetí vstupních dat proběhne série podmínek a je vybrána kategorie. Díky stromové struktuře je model velice rychlý. [9]



Obrázek 3: Jednoduchý rozhodovací strom, který identifikuje zvíře [10]

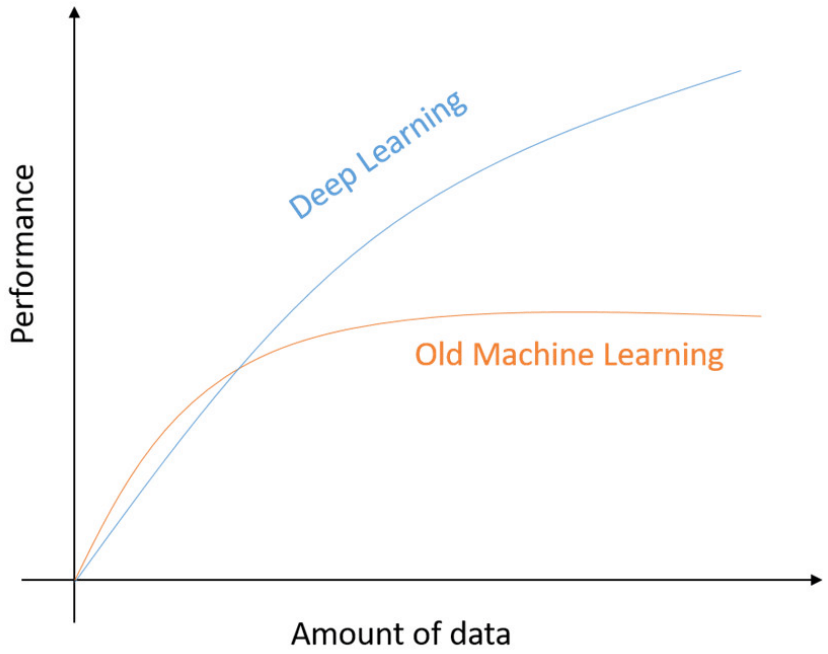
3.1.2 Náhodný les

Náhodný les je algoritmus klasifikace s dozorem, který obsahuje desítky, stovky až tisíce rozhodovacích stromů. Problém s jedním rozhodovacím stromem je to, že oproti ostatním metodám učení malá změna v datech může mít velký dopad na výsledek, proto jsou často nepřesné. Tento problém řeší právě náhodný les, který vytvoří skupinu stromů a u tvorby každého stromu náhodně vybere jeho příznaky. Potom už jen provede výpočty na všech stromech a vyhodnotí skupinu s největším počtem výskytů. Obrazně jde tato metoda popsat takto: Pokud položíte otázku z historie jednomu člověku, tak je velká pravděpodobnost, že odpoví špatně. Pokud stejnou otázku položíte sto lidem a vyberete nejčastější odpověď, tak je velká pravděpodobnost, že to bude právě ta správná odpověď. Náhodné lesy se dělí na klasifikační a regresní lesy. Pro

datování textů využijeme klasifikační les, protože budeme chtít data zařazovat do příslušného století. [9, 11]

3.2 Hluboké učení (Deep learning)

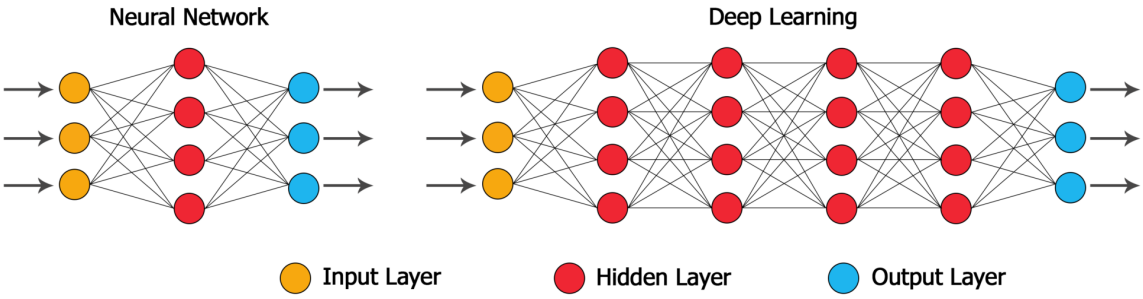
Hluboké učení je metoda strojového učení založená na neuronových sítích. Tato metoda je inspirovaná učením lidského mozku. Využití hlubokého učení při klasifikaci textu přináší potenciálně mnohem vyšší přesnost než tradiční algoritmy pro strojové učení. Je však nutno počítat s velkým vzorkem vstupních dat. [9]



Obrázek 4: Výkonnostní křivka hlubokého učení oproti starším metodám strojového učení [12]

Neurony jsou rozděleny do tří různých typů vrstev:

- Vstupní vrstva (Input layer) – Tato vrstva přijme vstupní data a předá je první skryté vrstvě.
- Skrytá vrstva (Hidden layer) – Skryté vrstvy provádějí matematické výpočty na vstupních datech. Na rozdíl od klasické neuronové sítě je v hlubokém učení vždy více než jedna skrytá vrstva.
- Výstupní vrstva (Output layer) – Vrací zpracované výstupní data.



Obrázek 5: Rozdíl mezi klasickou neuronovou sítí a hlubokým učením [13]

3.3 Přenesené učení (Transfer learning)

Přenesené učení je poměrně nová a výkonná oblast strojového učení, která má velký vliv na zpracování přirozeného jazyka. Dovoluje nám využít již předtrénované modely hlubokého učení pro specifické potřeby jako například klasifikaci textu nebo generování textu. Tyto modely jsou většinou trénovány metodou klasifikace bez dozoru na velkém obecném datasetu. S pomocí našich dat pak stačí model pouze 'dotrénovat' a přizpůsobit na náš dataset. Velká výhoda přeneseného učení je, že nepotřebujeme tolik dat jako například u hlubokého učení kde vytváříme model takzvaně 'od základů'. Nejznámější příklady takto předtrénovaných modelů jsou například StanfordNLP, ULMFiT nebo Transformer a BERT od Googlu. [14]

Konkrétně budeme využívat open-source metodu ULMFiT (Universal Language Model FIne-Tuning), který vytvořili Jeremy Howard (fast.ai) a Sebastian Ruder (DeepMind) a využívá model AWD-LSTM. ULMFiT může být využit na jakýkoli úkol v oblasti zpracování přirozeného jazyka. Je založen na datasetu 'Wikitext 103', který obsahuje zpracovanou část anglické wikipedie. [15, 16]

4 Vytvoření datasetu pro experimenty

Pro tvorbu datasetu jsem využil Project Gutenberg, což je rozsáhlá digitální knihovna, která obsahuje přes 60,000 eBooků. Výhodou Projektu Gutengerg je to, že je zcela zdarma, a to proto, že se zaměřuje na staré knihy, které již nejsou pod žádnou licencí. Problémem při tvorbě datasetu bylo to, že převážná většina prezentovaných knih je z 18. a 19. století a s každým vzdálenějším stoletím od těchto dvou se výběr knih výrazně zmenšoval. Z tohoto důvodu jsem se rozhodl dataset omezit na 17. až 20. století. Jednotlivé kategorie tedy jsou - 17. století, 18. století, 19. století a 20. století. Dataset tvoří dohromady téměř 100 částečně automaticky sesbíraných knih, kde z každé knihy je vybráno zhruba 400 vět a všechny testovací texty jsou v angličtině. Pro práci s textem byla využita sada knihoven NLTK, která je zaměřená na práci s lidským jazykem. Pro vektorizaci a převedení do binární podoby posloužila knihovna Keras. [17]

	text	century
0	I suppose we'll have all kinds of supplies thi...	20
1	When she coaxed and insisted, I repulsed her q...	20
2	I am very curious, I own: but I dare say the y...	19
3	Now then, what are you thinking of?	19
4	All I can do is to read to you, and I will do ...	19
...	...	...
38768	How could a guileless peasant steal in upon hi...	18
38769	This transaction had attracted the notice of s...	19
38770	Since my time.	19
38771	And thus, having found two or three broken oar...	18
38772	I'll go and observe your directions.	17

Obrázek 6: Ukázka prvních a posledních 5 záznamů v datasetu

4.1 Tokenizace

Tokenizace je proces rozdělení vět do pole, kde každý prvek pole tvoří jedno slovo věty. Tento process poskytuje dobrou přesnost a zároveň potřebuje méně výpočetního výkonu. [18]

```
rt = RegexpTokenizer(r'\w+')
dataset['text_token'] = dataset['text'].apply(lambda x: rt.tokenize(x.lower
    ()))
```

Třída RegexpTokenizer toto rozdělení provádí s pomocí regulárních výrazů. Zároveň s tím nám třída odstraní interpunkci. Text navíc ještě převedeme na malá písmena.

4.2 Odstranění stopslov

Stopslova neboli 'nevýznamové slova' jsou taková slova, která nám toho o podstatě textu moc neprozradí a vyskytují se ve všech textech. Tyto výrazy jsou proto při strojovém učení nechtěné. Většinou se jedná o spojky, předložky, booleovské operátory, velmi obecné a často se opakující výrazy. [19]

```
stopwords = nltk.corpus.stopwords.words('english')
dataset['text_nostop'] = dataset['text_token'].apply(lambda x: [w for w in x
    if w not in stopwords])
```

Z knihovny si načteme pole anglických stopslov a tyto slova vyfiltrujeme z datasetu.

4.3 Lemmatizace

Lemmatizace je proces, kdy je slovo převedeno do jeho základní podoby. Například na prvním řádku ve výsledném textu můžeme vidět převedení slova 'supplies' na 'supply'. Pro tyto účely je možné také využít stemizaci, ale lemmatizace má tu výhodu, že využívá přístup založený na slovníku, a proto je přesnější. [20]

```
wnl = nltk.WordNetLemmatizer()
dataset['text_lem'] = dataset['text_nostop'].apply(lambda x: [wnl.lemmatize(
    w) for w in x])
```

4.4 Vektorizace

Vektorizaci ve strojovém učení textů můžeme chápat jako kódování textu do numerické podoby tak, aby s tím algoritmus uměl pracovat. Existuje více způsobů jak aplikovat vektorizaci. U tohoto projektu využijeme vektorizaci pomocí slovníku - každé slovo, které bude využito musí být předem zaveden do slovníku. kde má každé slovo svoje číslo podle počtu výskytů. Tokenizované slova jsou pak nalezeny ve slovníku a jednoduše nahrazeny příslušným číslem. Nula je rezervovaný index, který je při vytváření slovníku vynechán. [18, 20]

```
t = Tokenizer()
t.fit_on_texts(dataset['text_lem'])
```

Funkce '`fit_on_texts`' má za úkol ze zadaného textu vytvořit a doplňovat slovník.

```
x_arr_tok = t.texts_to_sequences(dataset['text_lem'])
```

V tomto kroku funkce 'texts to sequences' nahrazuje slova za čísla ve slovníku.

```
max_length = max([len(s) for s in x_arr_tok])
x_train = pad_sequences(x_arr_tok, maxlen=max_length, padding='post')
```

Algoritmus potřebuje jednotnou délku vstupního pole, proto je potřeba zjistit délku nejdelšího pole a ostatní doplnit do této délky. O to se stará funkce '`pad_sequences`', která vyplní pole pomocí nul, jelikož těm nebylo přiděleno žádné slovo, a tak nemají žádný význam. A tímto je textová část datasetu hotová. [20]

	text	century	tokenization	no_stopwords	lemmatization
0	I suppose we'll have all kinds of supplies thi...	20	[i, suppose, we, ll, have, all, kinds, of, sup...	[suppose, kinds, supplies, winter, half, sunda...	[suppose, kind, supply, winter, half, sunday, ...
1	When she coaxed and insisted, I repulsed her q...	20	[when, she, coaxed, and, insisted, i, repulsed...	[coaxed, insisted, repulsed, quite, sternly]	[coaxed, insisted, repulsed, quite, sternly]
2	I am very curious, I own: but I dare say the y...	19	[i, am, very, curious, i, own, but, i, dare, s...	[curious, dare, say, young, lady, tell, tomorr...	[curious, dare, say, young, lady, tell, tomorr...
3	Now then, what are you thinking of?	19	[now, then, what, are, you, thinking, of]	[thinking]	[thinking]
4	All I can do is to read to you, and I will do ...	19	[all, i, can, do, is, to, read, to, you, and, ...	[read, till, heart, parts, interest, cream, book]	[read, till, heart, part, interest, cream, book]
...	...	...	...	...	...
38768	How could a guileless peasant steal in upon hi...	18	[how, could, a, guileless, peasant, steal, in,...	[could, guileless, peasant, steal, upon, lords...	[could, guileless, peasant, steal, upon, lords...
38769	This transaction had attracted the notice of s...	19	[this, transaction, had, attracted, the, notic...	[transaction, attracted, notice, ragged, urchi...	[transaction, attracted, notice, ragged, urchi...
38770	Since my time.	19	[since, my, time]	[since, time]	[since, time]
38771	And thus, having found two or three broken oar...	18	[and, thus, having, found, two, or, three, bro...	[thus, found, two, three, broken, oars, belong...	[thus, found, two, three, broken, oar, belongi...
38772	I'll go and observe your directions.	17	[i, ll, go, and, observe, your, directions]	[go, observe, directions]	[go, observe, direction]

Obrázek 7: Dataset po úprávách ze sady knihoven NLTK

4.5 Převod kategorií na binární pole

Nyní nám zbývá k textům vytvořit ještě pole s příslušnými kategoriemi. Pro metodu 'Náhodný les' nám stačí si jednoduše do pole přesunout sloupec kategorií (v tomto případě století).

```
for row in dataset['century']:
    y_train.append(row)
```

Pro Deep Learning je potřeba kategorie převést na binární pole. Tento krok je potřeba udělat kvůli pozdějšímu využití loss funkce 'categorical-crossentropy'. [20]

```
for row in dataset['century']:
    if int(row) == 17:
        y_train.append(0)
    if int(row) == 18:
        y_train.append(1)
    if int(row) == 19:
        y_train.append(2)
    if int(row) == 20:
        y_train.append(3)

y_train = to_categorical(y_train)
```

Pro lepší přehlednost si nejprve kategorie převedeme na čísla od 0 do 3. Následně na pole aplikujeme funkci '`to_categorical`', která nám kategorie převede na binární hodnoty. Pro příklad uvedu pole před kódováním [0, 1, 2, 3] a pole po kódování [[1. 0. 0. 0.], [0. 1. 0. 0.], [0. 0. 1. 0.], [0. 0. 0. 1.]].

4.6 Příprava dat pro přenesené učení

Zde pracujeme s předpřipraveným modelem ULMFiT, který je trénován na neupravovaných datech. S daty tedy není třeba dodatečně manipulovat jako v předchozích případech.

```
data_lm = TextLMDataBunch.from_df(train_df = dataset, valid_df =
    dataset_test, path = "")
data_clas = TextClasDataBunch.from_df(path = "", train_df = dataset,
    valid_df = dataset_test, vocab = data_lm.train_ds.vocab, bs = 32)
```

Fast.ai nabízí objekty TextLMDataBunch a TextClasDataBunch, které provedou veškerou přípravu pro využití s jazykovým modelem včetně vektorizace. Můžeme si všimnout, že `batch_size` (bs) se v tomto případě nastavuje už při zpracování dat.

5 Implementace několika vybraných metod

5.1 Náhodný les

5.1.1 Potřebné knihovny

Pro implementaci tohoto algoritmu jsem vybral open-source knihovnu SciKit-learn, což je jedna z nejpopulárnější knihoven pro strojové učení v jazyce python. Je postavena na třech neméně populárních knihovnách NumPy, SciPy a matplotlib. Knihovna pokrývá většinu algoritmů pro strojové učení. Tato knihovna pracuje pouze na CPU. [21]

5.1.2 Implementace

Teď když už máme vhodně zpracován dataset je implementace modelu náhodný les pomocí knihovny SciKit-learn poměrně jednoduchá.

```
random_forest = RandomForestClassifier(n_estimators = 50, max_depth = None,
    bootstrap = True, min_samples_split = 2)
```

V prvním kroku vytvoříme objekt klasifikátoru a specifikujeme vstupní hodnoty.

- `n_estimators` - Nastavuje počet stromů.
- `max_depth` - Hodnotou můžeme omezit maximální hloubku stromů. Pokud je nastavena hodnota 'None', tak je hloubka bez omezení.
- bootstrap - Pokud je tato hodnota nastavena na 'False', tak pro budování každého stromu je využit celý dataset.
- `min_samples_split` - Minimální počet vzorků potřebných pro rozdělení.

V druhém a posledním kroku zavoláme na model funkci 'fit', která model vytvoří na základě zpracovaného datasetu.

```
random_forest.fit(x_train, y_train)
```

- x - Pole s testovacími texty.
- y - Pole příslušných kategorií k textům.

5.2 Hluboké učení

5.2.1 Potřebné knihovny

Na implementaci hlubokého učení jsem zvolil vysokoúrovňovou knihovnu Keras, která je schopna běžet na knihovnách TensorFlow, CNTK nebo Theano. V tomto případě jsem vybral doporučenou knihovnu TensorFlow, který byl vyvinut týmem Google Brain pro interní účely, ale později byl zpřístupněn pro širokou veřejnost. Velkou výhodou TensorFlow je využití akcelerace pomocí GPU, což výrazně urychluje čas potřebný pro učení. [22, 20]

5.2.2 Implementace

Základní kámen hlubokého učení je model. Keras nabízí dva typy modelů - základní sekvenční a třídní API. Nám bude stačit základní sekvenční model, který reprezentuje lineární zásobník vrstev.

```
model = Sequential()
```

Jako první vrstvu využijeme 'Embedding'. Tato vrstva reprezentuje slova jako hustou vektorovou reprezentaci. Tohoto využívá například populární 'Word2Vec' nebo 'GloVe'.

```
model.add(Embedding(input_dim = vocab_size, output_dim = 750, input_length =
    max_length))
```

- `input_dim` - Velikost slovníku datasetu.
- `output_dim` - Každé slovo bude představovat vektor tolika dimenzí.
- `input_length` - Délka nejdelšího vstupního pole.

Nyní potřebujeme 'zarovnat' dvoudimenzionální výstup předešlé vrstvy do jednodimenzionální podoby, abychom umožnili spojení s poslední vrstvou. K tomuto slouží vrstva 'Flatten'.

```
model.add(Flatten())
```

'Dropout' je regularizační vrstva, která při učení náhodně vybírá a ignoruje neurony. Tímto vrstva dopomáhá k tomu, aby se model nezačal příliš vázat na dataset. [23]

```
model.add(Dropout(rate = 0.5))
```

- rate - Míra ignorovaných neuronů v procentech.

Poslední neboli výstupní vrstva reprezentuje nejzákladnější vrstvu v Kerasu - Dense.

```
model.add(Dense(units = 4, activation='softmax'))
```

- units - Počet kategorií ze kterých model vybírá (v tomto případě 4 století).
- activation - Volba aktivační funkce.

Oproti náhodnému lesu je zde navíc funkce 'compile', která provede výpočty na 'backend' úrovní (v tomto případě TensorFlow). V této funkci se také rozhoduje, zda účení bude probíhat na CPU, GPU nebo zároveň.

```
opt = Adam(lr = 0.0001)
model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics =
    ['accuracy'])
```

- loss - Loss funkce, která vyhodnocuje aktuální ztrátovost modelu.
- optimizer - Nastavení optimalizátoru.
- metrics - Metrika pro zhodnocení výsledků učení.

Funkce 'fit' vytvoří model na pevně daných iteracích datasetem.

```
model.fit(x = x_train, y = y_train, batch_size = 64, epochs = 8,
    validation_split = 0.1)
```

- x - Pole s testovacími texty.
- y - Pole příslušných kategorií k textům.
- `batch_size` - Počet řádků datasetu využitých při jednom trénování.
- epochs - Počet iterací při učení nad datasetem.
- `validation_split` - Specifikace části datasetu, která je vyhrazena pro testování a nezasahuje tak do učení.

Jako poslední funkce, kterou můžeme zavolat je funkce 'evaluate', která nám na konci učení vyhodnotí výsledky.

```
score, accuracy = model.evaluate(x = x_train, y = y_train, batch_size = 64)
```

Výsledné hodnoty si rovnou uložíme do proměnných.

- x - Pole s testovacími texty.
- y - Pole příslušných kategorií k textům.
- `batch_size` - Počet řádků datasetu využitých při jednom trénování.

5.3 Přenesené učení

5.3.1 Potřebné knihovny

Pro implementaci metody ULMFiT využijeme knihovnu fast.ai. Jedná se o open-source multifunkční knihovnu pro deep learning, která je založena na moderních a výzkumech osvědčených postupech. Backend pro tuto knihovnu zajišťuje velmi populárním framework pro strojové učení PyTorch. [24]

5.3.2 Implementace

Implementace se v tomto případě skládá ze dvou hlavních kroků. V prvním kroku musíme model přizpůsobit na naše data, a proto si vytvoříme a vytrénujeme model '`language_model_learner`'. Z prvního modelu potřebujeme pro klasifikaci pouze 'enkodér', což je část modelu, která obsahuje chápání vět. V druhém kroku využijeme enkodér pro klasifikaci skrze model '`text_classifier_learner`'.

```
learn = language_model_learner(data_lm, AWD_LSTM, drop_mult=0.5)
```

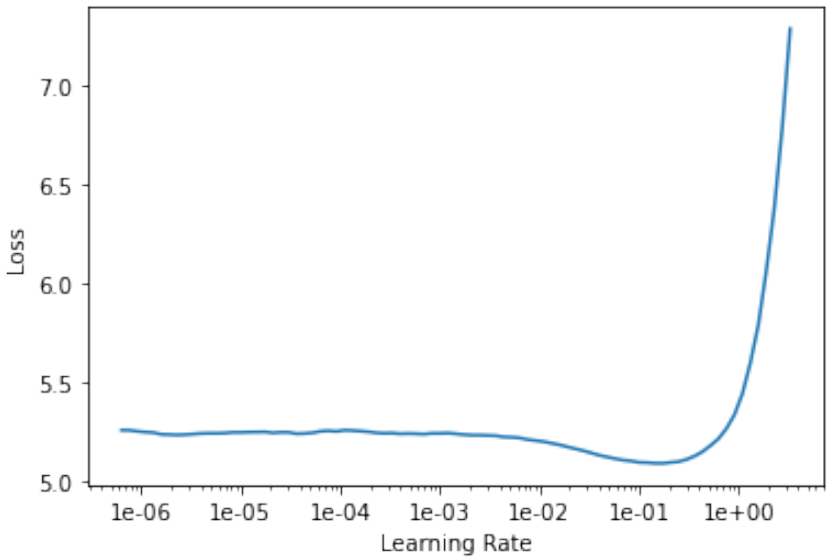
Jako první tedy vytvoříme objekt '`language_model_learner`' a zadáme potřebné parametry.

- 1.parametr - Předpřipravený dataset.
- 2.parametr - Model využitý pro učení.
- 3.parametr - Míra ignorovaných neuronů v procentech.

Míry učení je zde jeden z nejdůležitějších parametrů. Knihovna nabízí funkci '`lr_find`', která nám pomůže zjistit ideální learning rate pro začátek.

```
learn.lr_find()
learn.recorder.plot()
```

Z grafu rozsahu míry učení vidíme, že nejmenší hodnota je zhruba okolo míry učení 1e-1. Jako ideální míra učení se zde doporučuje posun o jednu jednotku směrem doleva. Učení tedy začneme s mírou učení 1e-2.



Obrázek 8: Graf rozsahu míry učení

Jak již bylo řečeno knihovna využívá moderní techniky v oblasti strojového učení. V tomto příkladu využijeme 'diskriminační doladění' a 'politiku jednoho cyklu'. Místo využití stejné míry učení (learning rate) na celý model je zde možnost využít na každou vrstvu jinou míru učení a to vždy po jednom cyklu. V tomto případě první cyklus s naší ideální mírou a druhý s jednou desetinou ideální míry učení. [15, 25]

```
learn.fit_one_cycle(1, 1e-2)
```

- 1.parametr - Počet cyklů.
- 2.parametr - Míra učení cyklu.

S pomocí prvního trénování jsme vytrénovali pouze poslední vrstvu modelu a zbytek jsme nechali nedotčen. Nyní pomocí metody 'unfreeze' model 'rozmrazíme' a druhý cyklus už bude trénovat celý model.

```
learn.unfreeze()
learn.fit_one_cycle(1, 1e-3)
```

Po druhém trénování máme model přizpůsobený na naše data a uložíme enkodér modelu pro následné využití u klasifikátoru.

```
learn.save_encoder("encoder")
```

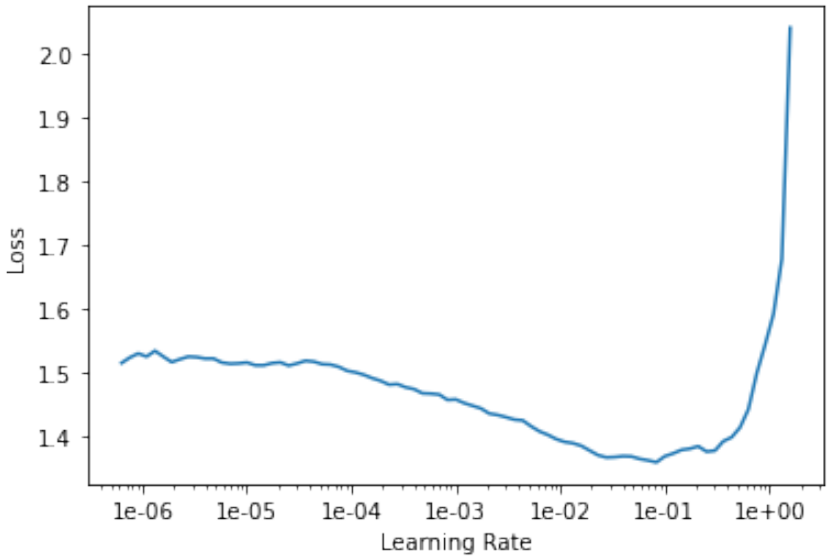
Nyní už máme vše potřebné pro vytvoření objektu '`text_classifier_learner`', který bude reprezentovat klasifikátor. Následně do něj načteme přizpůsobený enkodér.

```
learn = text_classifier_learner(data_clas, AWD_LSTM, drop_mult = 0.5)
learn.load_encoder("encoder")
```

- 1.parametr - Předpřipravený dataset.
- 2.parametr - Model využitý pro učení.
- 3.parametr - Míra ignorovaných neuronů v procentech.

Stejně jako v prvním případě chceme najít ideální míru učení.

```
learn.lr_find()
learn.recorder.plot()
```



Obrázek 9: Graf rozsahu míry učení klasifikátoru

Nejnižší hodnota se znovu pohybuje okolo hodnoty 1e-1, takže můžeme použít stejnou počáteční míru učení jako u prvního modelu.

```
learn.fit_one_cycle(1, 1e-2)
```

Po vytrénování poslední vrstvy 'rozmrazíme' dvě poslední vrstvy a model vytrénujeme postupně zase o něco více. Při trénování na více vrstvách je možné pomocí metody 'slice' změnit míru učení prvních a posledních vrstev.

```
learn.freeze_to(-2)
learn.fit_one_cycle(1, slice(5e-3/2., 5e-3))
```

Nakonec trénujeme na celém modelu s tím, že opět zmenšíme míru učení.

```
learn.unfreeze()
learn.fit_one_cycle(1, slice(2e-3/100, 2e-3))
```

5.4 Predikce

V této fázi máme model vytrénovaný a připravený na provedení predikce na datech.

```
y_predicted = model.predict(x_train_test)
```

Na model náhodného lesa zavoláme funkci 'predict' (v případě hlubokého učení je to funkce '`predict_classes`') a jako vstup zadáme pole testovacích vět, které chceme predikovat. Výstup si uložíme do pole, které bude obsahovat predikované třídy. U přeneseného učení predikujeme během učících cyklů.

5.5 Chybová matice

Při tvorbě klasifikačního modelu se nemůžeme spoléhat na celkovou přesnost v procentech a to obzvláště u multi-klasifikace. A to z toho důvodu, že model může vyhodnotit například 4 z 5 tříd na 100 procent a poslední na 0 procent. Přesnost by u tohoto modelu byla 80 procent, ale model by nefungoval správně, protože by nebyl schopen určovat všechny třídy. Proto se zde využívá tzv. chybová matice, která nám pomůže pochopit, zda je model schopen predikovat všechny třídy správně a také za co model zaměnil správný výsledek v případě chyby.

Pro pochopení se můžeme podívat na následující obrázek, kde je vyobrazena chybová matice binární klasifikace. Jsou zde 2 třídy - 'YES' (60 predikcí) a 'NO' (105 predikcí). Dohromady je zde 165 predikcí, kde na uhlopříčce vedoucí z levého horního okraje tabulku se nachází správně predikované hodnoty. Na druhém místě prvního řádku tabulky můžeme vidět hodnoty, které model chybně klasifikoval jako 'YES' a první pozici druhého řádku zase hodnoty, které model chybně klasifikoval jako 'NO'. [26]

Pro vytvoření a vykreslení matice do přehledné podoby využijeme knihovny scikit-learn, seaborn, pandas a matplotlib.

```
conf = confusion_matrix(y_trained_test, y_predicted)
```

Do chybové matice dosadíme na první pozici správně predikované třídy a následně třídy predikované modelem.

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Obrázek 10: Ukázka jednoduché chybové matice [26]

```
df_cm = pd.DataFrame(conf, [17, 18, 19, 20], [17, 18, 19, 20])
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt="d")
plt.show()
```

Vytvoříme 'DataFrame' knihovny pandas a načteme do něj matici a popisky ke třídám. Přes knihovnu seaborn vytvoříme z dat obrázek matice a pomocí knihovny matplotlib si obrázek zobrazíme.

A nyní máme vše potřebné pro správné zhodnocení modelů.

6 Vyhodnocení výsledků

Při učení občas nastává situace, kdy se model až moc přizpůsobí datasetu a proto je vhodné model testovat na zcela nových datech mimo základní dataset pro relevantní zhodnocení. K těmto účelům jsem si vytvořil testovací dataset, který obsahuje vzorek jedné knihy z každé kategorie (zhruba 400 vět na knihu). Na tento testovací dataset musí být samozřejmě aplikovány stejné zpracovávací metody textu jako na původní dataset.

6.1 Náhodný les

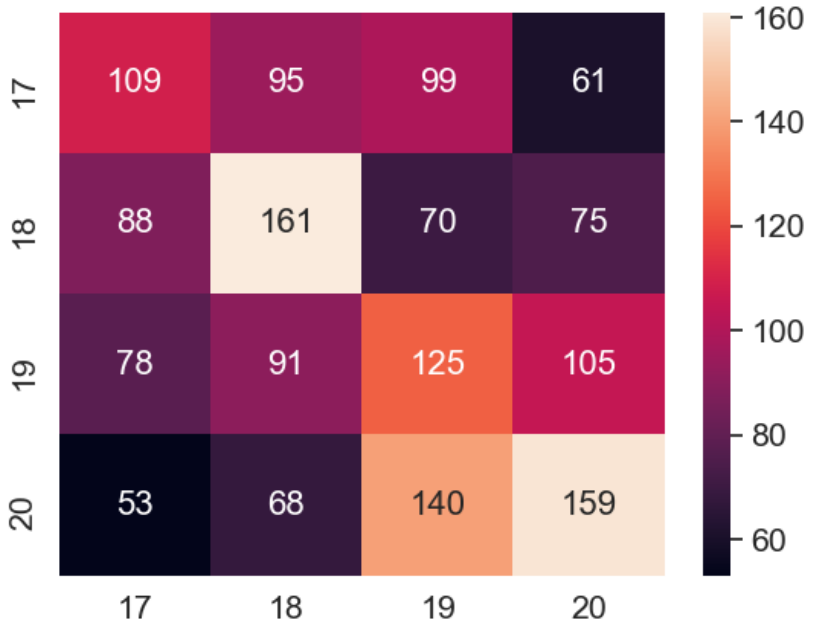
Ve strojovém učení neexistuje ideální rozložení volitelných parametrů, proto je třeba parametry otestovat například experimentálně a dojít k nejideálnější konfiguraci pro daný dataset. Níže je vidět tabulka testovaných konfigurací modelu s výstupní hodnotou přesnosti a počtu správně klasifikovaných tříd modelem.

Tabulka 1: Experimentální testování modelu náhodný les

n_estimators	**max_depth**	**min_samples_split**	**přesnost**	**klasifikované třídy**
10	10	2	36%	2
10	20	2	32%	2
20	20	2	36%	2
30	None	2	35%	3
50	None	3	35%	2
50	**None**	**2**	**37%**	**4**
75	None	2	36%	3
100	None	4	37%	2
200	None	2	36%	2
200	None	3	38%	2
350	None	2	39%	2

Z tabulky jde vidět, že pouze konfigurace s 50 podstromy, minimálním počtem pro rozdělení 2 a neomezenou hloubkou klasifikovala správně všechny čtyři třídy. Při dalším přidáváním podstromů sice model nabíral na přesnosti, ale postupně více a více vět začal řadit pouze do dvou tříd.

Výsledný model byl vytrénován za 5,5 sekundy.



Obrázek 11: Chybová matice modelu náhodného lesa

V matici můžeme vidět, že model sice správně klasifikoval všechny třídy, ale u 17. století model klasifikoval 99 vzorků chybně do 19. století. U 17. století je tak rozdíl mezi druhou nejpočetnější skupinou pouze o 10 vzorků. Naopak 18. století model rozpoznal s dostatečnou rezervou.

6.2 Hluboké učení

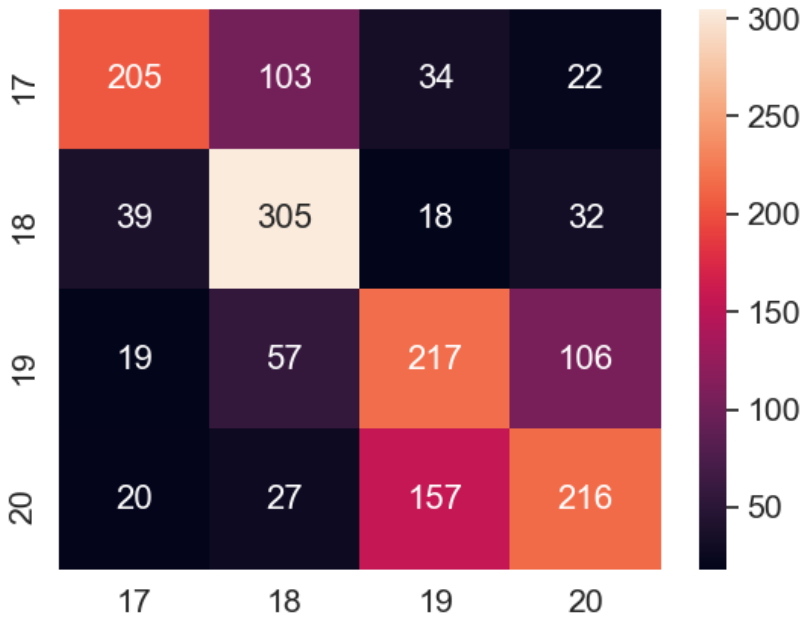
Níže je vidět tabulka testovaných konfigurací modelu s výstupní hodnotou přesnosti a počtu správně klasifikovaných tříd modelem. U hlubokého učení testujeme konfigurace změnou parametrů počtu epoch, velikostí batche a počtem dimenzí Embedding vrstvy.

Tabulka 2: Experimentální testování modelu hlubokého učení

epochs	batch_size	output_dim	přesnost	klasifikované třídy	nejhorší třída
2	16	50	46%	2	9
2	16	150	52%	2	53
2	32	150	51%	2	75
4	32	250	58%	4	185
4	32	350	60%	4	158
6	32	450	58%	4	182
6	64	600	59%	4	175
8	**64**	**750**	**59%**	**4**	**205**
8	32	750	58%	4	198
8	64	850	58%	4	189
16	64	750	57%	4	176
32	64	750	54%	4	156

Z tabulky je jasné, že hluboké učení dokáže klasifikovat všechny třídy téměř ve všech případech. Vybral jsem tedy konfiguraci, ve které byla nejpočetnější třída s nejméně správně predikovanými vzorky.

Výsledný model byl vytrénován za 101 sekund.



Obrázek 12: Chybová matice modelu hluboké učení

Matice už vypadá mnohem lépe. Všechny třídy jsou predikované správně s dostatečnou rezervou. Nejhůře dopadlo 17. století, které mělo 205 správně klasifikovaných vzorků.

6.3 Přenesené učení

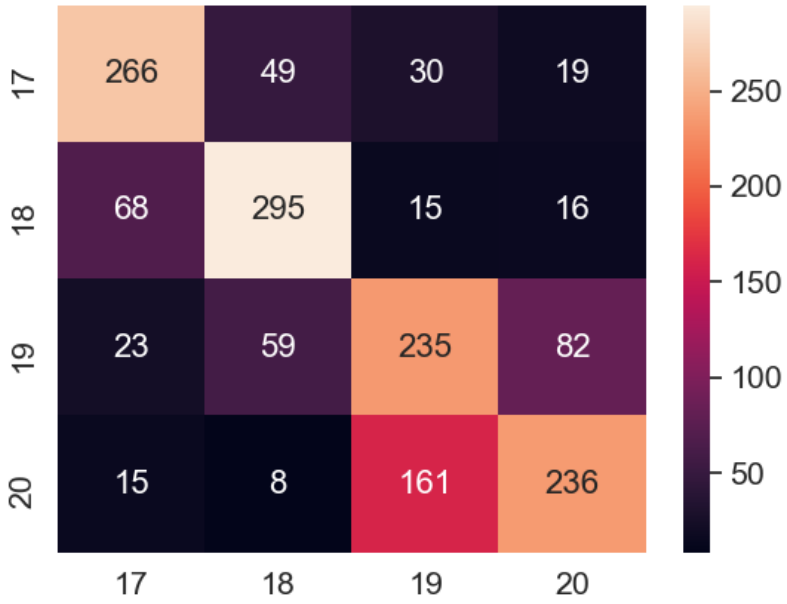
U experimentálního testování přeneseného učení jsou jako vstupní parametry velikost batche a dropout. Jako výstupní parametry opět přesnost, správně klasifikované třídy a třída s nejméně správně klasifikovanými vzorky.

Tabulka 3: Experimentální testování modelu přeneseného učení

batch_size	dropout	přesnost	klasifikované třídy	nejhorší třída
8	0.5	66%	4	209
16	0.5	64%	4	224
32	**0.5**	**65%**	**4**	**235**
64	0.5	64%	4	206
32	0.7	63%	4	202
32	0.3	64%	4	203

Stejně jako u hlubokého učení zde nelze vybrat konfiguraci podle počtu klasifikovaných tříd. Jako ideální možnost zde však vyčnívá konfigurace s velikostí batche 32 a dropoutem 50 procent, která byla schopna dosáhnout přesnosti 65 procent, ale hlavně nějvyšší hodnoty u nejhorší třídy a to konkrétně 235.

Výsledný model byl vytrénován za 4 hodiny a 4 minuty.



Obrázek 13: Chybová matice přeneseného učení

Jako jediný problém chybové matice bychom mohli považovat 161 vzorků 20. století, které jsou chybně klasifikované jako 19. století. Rozdíl hodnot je ale stále dost velký.

6.4 Vyhodnocení

Tabulka 4: Přehled nejlepších konfigurací modelů

Metoda	**Čas učení [s]**	**přesnost**	**klasifikované třídy**	**nejhorší třída**
Náhodný les	5,5	37%	4	109
Hluboké učení	101	59%	4	205
Přenesené učení	14 640	65%	4	235

Metoda náhodný les založená na podmínkách sice byla schopna v určité konfiguraci zařadit všechny čtyři knihy správně, ale s rezervou na kterou by se v praxi rozhodně nedalo spoléhat. Naproti tomu model hlubokého učení přinesl obrovský skok v přesnosti a rezervě oproti špatně predikovaným vzorkům. Zajímavé je, že pro hluboké učení nepřinášelo zvýšení učících cyklů žádné další výsledky. Model tak při použití grafické akcelerace lze na tomto příkladu vytrénovat rychle. To je pravděpodobně dáno tím, že v datech je jasnější rozpoznávací vzor, a tak je rychle objeven. I když datování textu není nejlepší příklad využití u přeneseného učení, tak navýšilo přesnost modelu o šest procent, což není tak velký skok jako u hlubokého učení, ale i přesto vylepšilo model vybalancováním a navýšením správně predikovaných tříd. Největší slabinou modelů jsou jasně vzorky 20. století, které byly klasifikovány jako 19. století a to nejspíš proto, že valná většina knih z 20. století na projektu Gutenberg bylo z první poloviny 20. století. Pokud bychom se podívali na metriku správně uhodnutých knih, tak jsou zde modely i po testování na další sadě čtyř knih téměř stoprocentní (model náhodný les při hlubším testování nedokázal zařadit jednu knihu 19. století).

7 Závěr

Tato bakalářská práce měla za cíl vytvořit vhodný vzorek dat a následně ho využít pro dataci textu do správného období. Pro tento úkol bylo potřeba vytvořit klasifikační systém, který bude podle zvolených dat predikovat správné časové období.

V teoretické části byly popsány tři možné přístupy pro tvorbu právě takového klasifikačního systému a následně tři konkrétní metody metody strojového učení, které je možné využít pro správnou klasifikaci textu. Jedná se metody náhodný les, hluboké učení a přenesené učení. Praktická část práce začíná připravením datasetu pro tvorbu modelů. Kapitola se však převážně zabývá úpravou dat, protože pro využití každé z metod je potřeba data jinak upravit. Následuje implementace jednotlivých zvolených metod. Každá z metod je implementována pomocí jiných knihoven, ale všechny zůstávají v jazyce python. Pro správné vyhodnocení výsledků klasifikace bylo ještě potřeba naimplementovat kód pro vygenerování a zobrazení chybové matice. Poslední kapitola obsahuje vyhodnocení jednotlivých metod a celkové shrnutí výsledků.

S dosaženým výsledkem jsem spokojen. Náhodný les sice výrazně zaostával, ale metody hlubokého a přeneseného učení dosahovali velice dobrých výsledků na testovaných datech. Chtěl bych podotknout, že přesnost s datovým vzorkem vět nebude nejspíš nikdy úplně stoprocentní, protože texty všech století obsahují některé velice obecné věty, které bychom mohli najít v jakékoliv uvedené kategorii. Obecně se tedy dá říct, že čím více testovacích dat se modelům poskytne, tím větší je pravděpodobnost správné predikce období.

Pokud bychom se bavili o reálném využití modelů v praxi, tak nad zmíněným datasetem nejspíše nenajde velké využití, protože století od 17. do 20. jsou velice dobře zmapované. K reálnému využitím by byl potřeba sestavit dataset z mnohem staršího období, což by však vyžadovalo mnohem více času na sesbírání takových dat.

Programovací jazyk python se ukázal být jako správná volba a to hlavně kvůli jeho širokému využití v oblasti strojového učení.

Literatura

1. *Rule-Based System* [online] [cit. 2020-02-10]. Dostupné z: https://www.sciencedirect.com/topics/engineering/rule-based-system.
2. *What is a rule-based system? What is it not?* [online] [cit. 2020-02-10]. Dostupné z: https://www.thinkautomation.com/eli5/what-is-a-rule-based-system-what-is-it-not/.
3. *Machine Learning - What it is and why it matters* [online] [cit. 2020-02-10]. Dostupné z: https://www.sas.com/en_us/insights/analytics/machine-learning.html/.
4. *What is Machine Learning? A definition* [online] [cit. 2020-02-10]. Dostupné z: https://expertsystem.com/machine-learning-definition/.
5. *How to Build a Better Machine Learning Pipeline* [online] [cit. 2020-04-10]. Dostupné z: https://www.datanami.com/2018/09/05/how-to-build-a-better-machine-learning-pipeline/.
6. *Learning to classify text* [online] [cit. 2020-02-10]. Dostupné z: http://www.nltk.org/book/ch06.html/.
7. *Examples of supervised learning* [online] [cit. 2020-03-18]. Dostupné z: https://www.researchgate.net/figure/Examples-of-Supervised-Learning-Linear-Regression-and-Unsupervised-Learning_fig3_336550812.
8. *Hybrid Approach Combining Machine Learning and a Rule-Based Expert System for Text Categorization.* [online] [cit. 2020-02-10]. Dostupné z: https://www.researchgate.net/publication/221438964_Hybrid_Approach_Combining_Machine_Learning_and_a_Rule-Based_Expert_System_for_Text_Categorization/.
9. AGGARWAL, Charu C. *Machine learning for text.* New York, NY: Springer Science+Business Media, 2018. ISBN 978-3-319-73530-6.
10. *Machine Learning Algorithms : Decision Trees* [online] [cit. 2020-03-18]. Dostupné z: https://mc.ai/machine-learning-algorithms-decision-trees/.
11. *Random Forest: simple explanation* [online] [cit. 2020-03-18]. Dostupné z: https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d.
12. *The performance of deep learning with respect to the amount of data.* [online] [cit. 2020-04-10]. Dostupné z: https://www.researchgate.net/figure/The-performance-of-deep-learning-with-respect-to-the-amount-of-data_fig3_331540139.
13. *Neural networks and deep learning and overview* [online] [cit. 2020-03-18]. Dostupné z: https://alphabold.com/neural-networks-and-deep-learning-an-overview/.

14. *Transfer learning and the art of using Pre-trained Models in Deep Learning* [online] [cit. 2020-03-30]. Dostupné z: https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/.

15. HOWARD, Jeremy; RUDER, Sebastian. Fine-tuned Language Models for Text Classification. *CoRR*. 2018, roč. abs/1801.06146. Dostupné z arXiv: 1801.06146.

16. *Efficient multi-lingual language model fine-tuning* [online] [cit. 2020-03-30]. Dostupné z: https://nlp.fast.ai/.

17. *Project Gutenberg* [online] [cit. 2020-03-16]. Dostupné z: https://www.gutenberg.org/.

18. *Google Developers - Step 3 Prepare Your Data* [online] [cit. 2020-03-16]. Dostupné z: https://developers.google.com/machine-learning/guides/text-classification/step-3.

19. *Databáze Národní knihovny ČR* [online] [cit. 2020-03-16]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000665&local_base=KTD.

20. *Home - Keras Documentation* [online] [cit. 2020-03-16]. Dostupné z: http://keras.io/.

21. *SciKit learn - machine learning in Python* [online] [cit. 2020-03-18]. Dostupné z: https://scikit-learn.org/.

22. *TensorFlow* [online] [cit. 2020-03-18]. Dostupné z: https://www.tensorflow.org/.

23. *Dropout Regularization in Deep Learning Models With Keras* [online] [cit. 2020-03-19]. Dostupné z: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/.

24. *Fastai documentation* [online] [cit. 2020-03-30]. Dostupné z: https://docs.fast.ai/.

25. *Finding Good Learning Rate and The One Cycle Policy.* [online] [cit. 2020-03-30]. Dostupné z: https://towardsdatascience.com/finding-good-learning-rate-and-the-one-cycle-policy-7159fe1db5d6.

26. *Simple guide to confusion matrix terminology* [online] [cit. 2020-03-19]. Dostupné z: https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/.

Přílohy

1. Bakalářská práce - obsah práce ve formátu PDF
2. Náhodný les - soubor se zdjovým kódem, datasetem a předpřipraveným modelem.
3. Hluboké učení - soubor se zdjovým kódem, datasetem a předpřipraveným modelem.
4. Přenesené učení - soubor se zdjovým kódem, datasetem, předpřipraveným modelem a souborem '.ipynb' ve formátu jupyter notebooku s průběhem učení.